1. What index value does the third element of an array have?

The **third element** has an index of **2**, because array indexes start at **0**.

---

2. Write the declaration for an array named `quantities` that stores 20 integers.

```
int[] quantities = new int[20];
```

---

3. Write a declaration for an array named `heights` storing the numbers 1.65, 2.15, and 4.95.

```
double[] heights = {1.65, 2.15, 4.95};
```

---

4. Write a for-each statement that displays the integer values stored in an array named `grades`.

```
for (int g : grades) {
    System.out.println(g);
}
```

---

6. How does passing an entire array to a method differ from passing a single element of the array?

- Passing **a single element** sends **just that value** (a copy of it).
- Passing **the entire array** sends a **reference** to the array, meaning the method can **change the actual array**.

---

7. Why are offset array indexes required in some cases?

Offset indexes are used when you need to:

- Start at a **specific position** in the array
- Compare two arrays with **different starting points**
- Skip elements
- Access an index relative to another (like `i + 1` or `i - 1`)

They help match data positions or perform calculations that depend on relative positions.

---

8. What output is displayed by the statements below?

```
String name = "Elaine";

System.out.println(name.charAt(3));
```

Answer:

Index 3 in `"Elaine"` is the letter **'i'**.

Output:

```
i
```

---

10. Give an example of when a dynamic array might be a better structure choice over an array.

A **dynamic array** (like `ArrayList`) is better when:

- You don't know how many items you will need to store
- The number of items will **change during the program**
- You frequently need to **add or remove** elements

Example:
Storing names entered by a user until they type "stop".

---

11. How does the ArrayList `indexOf()` method determine equality between the object passed to the method and an element in the array?

`indexOf()` uses the object's **equals()** method to check if two objects are considered equal.
If `equals()` returns **true**, it counts as a match.

---

12. How can the values of wrapper class objects be compared?

Wrapper objects (like `Integer`, `Double`) can be compared using:

1. `.equals()` — compares values

```
Integer a = 5;
Integer b = 5;
a.equals(b);   // true
```

2. Comparing primitive values (auto-unboxing)

```
a == b;   // also true in many cases because they become int values
```